prance Documentation

Release 0.22.11.4.1.dev3+g225635a

Jens Finkhaeuser

Jan 07, 2023

CONTENTS

1 Usage **3**
- 1.1 Installation . . . 3
- 1.2 Command Line Interface . . . 3
- 1.3 Code . . . 4
- 1.4 Compatibility . . . 4
- 1.5 Partial Reference Resolution . . . 5
- 1.6 Extensions . . . 6

2 Contributing **7**

3 License **9**

4 API/Modules **11**
- 4.1 prance . . . 11
- 4.2 prance.mixins . . . 13
- 4.3 prance.convert . . . 14
- 4.4 prance.util.formats . . . 16
- 4.5 prance.util.fs . . . 18
- 4.6 prance.util.iterators . . . 20
- 4.7 prance.util.resolver . . . 21
- 4.8 prance.util.url . . . 21
- 4.9 prance.util.exceptions . . . 23
- 4.10 prance.util.path . . . 23

5 Changes **25**

6 Prance 0.22.11.04.0 **27**
- 6.1 Features . . . 27

7 Prance 0.21.8.0 (2021-08-06) **29**
- 7.1 Features . . . 29
- 7.2 Bugfixes . . . 29

8 Prance 0.21.2 (2021-05-18) **31**
- 8.1 Bugfixes . . . 31

9 v0.21.1 (2021-05-18) **33**

10 v0.21.0 (2021-05-18) **35**
- 10.1 Features . . . 35

11 v0.20.2 37

12 v0.20.1 39

13 v0.20.0 41

14 v0.19.0 43

15 v0.18.3 45

16 v0.18.2 47

17 v0.18.1 49

18 v0.17.0 51

19 v0.16.2 53

20 v0.16.1 55

Python Module Index 57

Index 59

Swagger/OpenAPI 2.0 Parser for Python

Prance provides parsers for Swagger/OpenAPI 2.0 and 3.0 API specifications in Python. It uses openapi_spec_validator, swagger_spec_validator or flex to validate specifications, but additionally resolves JSON references in accordance with the OpenAPI spec.

Mostly the latter involves handling non-URI references; OpenAPI is fine with providing relative file paths, whereas JSON references require URIs at this point in time.

CHAPTER

ONE

USAGE

1.1 Installation

Prance is available from PyPI, and can be installed via pip:

```
$ pip install prance
```

Note that this will install the code, but additional subpackages must be specified to unlock various pieces of functionality. At minimum, a parsing backend must be installed. For the CLI functionality, you need further dependencies.

The recommended installation installs the CLI, uses ICU and installs one validation backend:

```
$ pip install prance[osv,icu,cli]
```

Make sure you have ICU Unicode Library installed, as well as Python dev library before running the commands above. If not, use the following commands:

```
$ sudo apt-get install libicu-dev python3-dev # Ubuntu/Debian
$ sudo dnf install libicu-devel python3-devel # Fedora
```

1.2 Command Line Interface

After installing prance, a CLI is available for validating (and resolving external references in) specs:

```
# Validates with resolving
$ prance validate path/to/swagger.yml

# Validates without resolving
$ prance validate --no-resolve path/to/swagger.yml

# Fetch URL, validate and resolve.
$ prance validate http://petstore.swagger.io/v2/swagger.json
Processing "http://petstore.swagger.io/v2/swagger.json"...
 -> Resolving external references.
Validates OK as Swagger/OpenAPI 2.0!
```

Validation is not the only feature of prance. One of the side effects of resolving is that from a spec with references, one can create a fully resolved output spec. In the past, this was done via options to the `validate` command, but now there's a specific command just for this purpose:

```
# Compile spec
$ prance compile path/to/input.yml path/to/output.yml
```

Lastly, with the arrival of OpenAPI 3.0.0, it becomes useful for tooling to convert older specs to the new standard. Instead of re-inventing the wheel, prance just provides a CLI command for passing specs to the web API of swagger2openapi - a working internet connection is therefore required for this command:

```
# Convert spec
$ prance convert path/to/swagger.yml path/to/openapi.yml
```

1.3 Code

Most likely you have spec file and want to parse it:

```
from prance import ResolvingParser
parser = ResolvingParser('path/to/my/swagger.yaml')
parser.specification  # contains fully resolved specs as a dict
```

Prance also includes a non-resolving parser that does not follow JSON references, in case you prefer that.

```
from prance import BaseParser
parser = BaseParser('path/to/my/swagger.yaml')
parser.specification  # contains specs as a dict still containing JSON references
```

On Windows, the code reacts correctly if you pass posix-like paths (`/c:/swagger`) or if the path is relative. If you pass absolute windows path (like `c:\swagger.yaml`), you can use `prance.util.fs.abspath` to convert them.

URLs can also be parsed:

```
parser = ResolvingParser('http://petstore.swagger.io/v2/swagger.json')
```

Largely, that's it. There is a whole slew of utility code that you may or may not find useful, too. Look at the full documentation for details.

1.4 Compatibility

Python Versions

Version 0.16.2 is the last version supporting Python 2. It was released on Nov 12th, 2019. Python 2 reaches end of life at the end of 2019. If you wish for updates to the Python 2 supported packages, please contact the maintainer directly.

Until fairly recently, we also tested with PyPy. Unfortunately, Travis isn't very good at supporting this. So in the absence of spare time, they're disabled. Issue 50 tracks progress on that.

Similarly, but less critically, Python 3.4 is no longer receiving a lot of love from CI vendors, so automated builds on that version are no longer supported.

Backends

Different validation backends support different features.

Backend	Python Version	OpenAPI Version	Strict Mode	Notes	Available From	Link
swagger-spec-validator	2 and 3	2.0 only	yes	Slow; does not accept integer keys (see strict mode).	prance 0.1	swagger_spec_validator
flex	2 and 3	2.0 only	n/a	Fastest; unfortunately deprecated.	prance 0.8	flex
openapi-spec-validator	2 and 3	2.0 and 3.0	yes	Slow; does not accept integer keys (see strict mode).	prance 0.11	openapi_spec_validator

You can select the backend in the constructor of the parser(s):

```
parser = ResolvingParser('http://petstore.swagger.io/v2/swagger.json', backend =
↪'openapi-spec-validator')
```

No backend is included in the dependencies; they are detected at run-time. If you install them, they can be used:

```
$ pip install openapi-spec-validator
$ pip install prance
$ prance validate --backend=openapi-spec-validator path/to/spec.yml
```

A note on flex usage: While flex is the fastest validation backend, unfortunately it is no longer maintained and there are issues with its dependencies. For one thing, it depends on a version of *PyYAML* that contains security flaws. For another, it depends explicitly on older versions of *click*.

If you use the flex subpackage, therefore, you do so at your own risk.

Compatibility

See COMPATIBILITY.rst for a list of known issues.

1.5 Partial Reference Resolution

It's possible to instruct the parser to only resolve some kinds of references. This allows e.g. resolving references from external URLs, whilst keeping local references (i.e. to local files, or file internal) intact.

```
from prance import ResolvingParser
from prance.util.resolver import RESOLVE_HTTP

parser = ResolvingParser('/path/to/spec', resolve_types = RESOLVE_HTTP)
```

Multiple types can be specified by OR-ing constants together:

```
from prance import ResolvingParser
from prance.util.resolver import RESOLVE_HTTP, RESOLVE_FILES

parser = ResolvingParser('/path/to/spec', resolve_types = RESOLVE_HTTP | RESOLVE_FILES)
```

1.6 Extensions

Prance includes the ability to reference outside swagger definitions in outside Python packages. Such a package must already be importable (i.e. installed), and be accessible via the ResourceManager API (some more info here).

For example, you might create a package `common_swag` with the file `base.yaml` containing the definition

```
definitions:
  Severity:
    type: string
    enum:
    - INFO
    - WARN
    - ERROR
    - FATAL
```

In the `setup.py` for `common_swag` you would add lines such as

```
packages=find_packages('src'),
package_dir={'': 'src'},
package_data={
    '': '*.yaml'
}
```

Then, having installed `common_swag` into some application, you could now write

```
definitions:
  Message:
    type: object
    properties:
      severity:
        $ref: 'python://common_swag/base.yaml#/definitions/Severity'
      code:
        type: string
      summary:
        type: string
      description:
        type: string
    required:
    - severity
    - summary
```

CHAPTER

TWO

CONTRIBUTING

See CONTRIBUTING.md for details.

Professional support is available through finkhaeuser consulting.

CHAPTER

THREE

LICENSE

Licensed under MIT. See the LICENSE.txt file for details.

“Prancing unicorn” logo image Copyright (c) Jens Finkhaeuser. Made by Moreven B. Use of the logo is permitted under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International license.

CHAPTER

FOUR

API/MODULES

prance	Prance implements parsers for Swagger/OpenAPI 2.0 and 3.0.0 API specs.
prance.mixins	Defines Mixins for parsers.
prance.convert	Functionality for converting from Swagger/OpenAPI 2.0 to OpenAPI 3.0.0.
prance.util.formats	This submodule contains file format related utility code for Prance.
prance.util.fs	This submodule contains file system utilities for Prance.
prance.util.iterators	This submodule contains specialty iterators over specs.
prance.util.resolver	This submodule contains a JSON inlining reference resolver.
prance.util.url	This submodule contains code for fetching/parsing URLs.
prance.util.exceptions	This submodule contains helpers for exception handling.
prance.util.path	This module contains code for accessing values in nested data structures.

4.1 prance

Prance implements parsers for Swagger/OpenAPI 2.0 and 3.0.0 API specs.

See https://openapis.org/ for details on the specification.

Included is a BaseParser that reads and validates swagger specs, and a ResolvingParser that additionally resolves any $ref references.

4.1.1 Exceptions

exception prance.**ValidationError**

Bases: Exception

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

4.1.2 Classes

class prance.**BaseParser**(*url=None*, *spec_string=None*, *lazy=False*, ***kwargs*)

Bases: *YAMLMixin*, *JSONMixin*

The BaseParser loads, parses and validates OpenAPI 2.0 and 3.0.0 specs.

Uses `YAMLMixin` and `JSONMixin` for additional functionality.

json()

Return a JSON representation of the specifications.

Returns
JSON representation.

Return type
dict

parse()

When the BaseParser was lazily created, load and parse now.

You can use this function to re-use an existing parser for parsing multiple files by setting its url property and then invoking this function.

specs_updated()

Test if self.specficiation changed.

Returns
Whether the specs changed.

Return type
bool

yaml()

Return a YAML representation of the specifications.

Returns
YAML representation.

Return type
dict

BACKENDS = {'flex': ((2,), '_validate_flex'), 'openapi-spec-validator': ((2, 3), '_validate_openapi_spec_validator'), 'swagger-spec-validator': ((2,), '_validate_swagger_spec_validator')}

SPEC_VERSION_2_PREFIX = 'Swagger/OpenAPI'

SPEC_VERSION_3_PREFIX = 'OpenAPI'

class prance.**ResolvingParser**(*url=None*, *spec_string=None*, *lazy=False*, ***kwargs*)

Bases: *BaseParser*

The ResolvingParser extends BaseParser with resolving references by inlining.

json()

Return a JSON representation of the specifications.

Returns
JSON representation.

Return type
dict

`parse()`

When the BaseParser was lazily created, load and parse now.

You can use this function to re-use an existing parser for parsing multiple files by setting its url property and then invoking this function.

`specs_updated()`

Test if self.specficiation changed.

Returns
Whether the specs changed.

Return type
bool

`yaml()`

Return a YAML representation of the specifications.

Returns
YAML representation.

Return type
dict

```
BACKENDS = {'flex':  ((2,), '_validate_flex'), 'openapi-spec-validator':  ((2, 3),
'_validate_openapi_spec_validator'), 'swagger-spec-validator':  ((2,),
'_validate_swagger_spec_validator')}
```

```
SPEC_VERSION_2_PREFIX = 'Swagger/OpenAPI'
```

```
SPEC_VERSION_3_PREFIX = 'OpenAPI'
```

4.2 prance.mixins

Defines Mixins for parsers.

The Mixins are here mostly for separation of concerns.

4.2.1 Classes

`class prance.mixins.CacheSpecsMixin`

Bases: `object`

CacheSpecsMixin helps determine if self.specification changed.

It does so by caching a shallow copy on-demand.

`specs_updated()`

Test if self.specficiation changed.

Returns
Whether the specs changed.

Return type
bool

class prance.mixins.**JSONMixin**

Bases: *CacheSpecsMixin*

JSONMixin returns a JSON representation of the specification.

It uses *CacheSpecsMixin* for lazy evaluation.

json()

Return a JSON representation of the specifications.

Returns
JSON representation.

Return type
dict

specs_updated()

Test if self.specficiation changed.

Returns
Whether the specs changed.

Return type
bool

class prance.mixins.**YAMLMixin**

Bases: *CacheSpecsMixin*

YAMLMixin returns a YAML representation of the specification.

It uses *CacheSpecsMixin* for lazy evaluation.

specs_updated()

Test if self.specficiation changed.

Returns
Whether the specs changed.

Return type
bool

yaml()

Return a YAML representation of the specifications.

Returns
YAML representation.

Return type
dict

4.3 prance.convert

Functionality for converting from Swagger/OpenAPI 2.0 to OpenAPI 3.0.0.

The functions use https://mermade.org.uk/ APIs for conversion.

4.3.1 Exceptions

exception prance.convert.**ConversionError**

Bases: ValueError

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

4.3.2 Functions

prance.convert.**convert_spec**(*parser_or_spec*, *parser_klass=None*, **args*, ***kwargs*)

Convert an already parsed spec to OpenAPI 3.x.y.

Returns a new parser instance with the parsed specs, if possible.

The first parameter is either a parsed OpenAPI 2.0 spec, or a parser instance, i.e. something derived from prance.BaseParser. If a parser, the returned parser's options are taken from this source parser.

If the first parameter is a parsed spec, you must specify the class of parser to instantiate. You can specify other options as key word arguments. See the parser klass for details.

Any key word arguments specified here also override options from a source parser.

This parametrization may seem a little convoluted. What it does, though, is allow maximum flexibility. You can create parsed (but unvalidated) OpenAPI 3.0 specs even if you only have backends that support version 2.0. You can pass the source parser, and the lazy flag, and that's it. If your version 2.0 specs were valid, there's a good chance your converted 3.0 specs are also valid.

Parameters

- **parser_or_spec** (*mixed*) – A dict (spec) or an instance of BaseParser
- **parser_klass** (*type*) – [optional] A parser class to instantiate for the result.

Returns
A parser instance.

Return type
BaseParser or derived.

prance.convert.**convert_str**(*spec_str*, *filename=None*, ***kwargs*)

Convert the serialized spec.

We parse the spec first to ensure there is no parse error, then send it off to the API for conversion.

Parameters

- **spec_str** (*str*) – The specifications as string.
- **filename** (*str*) – [optional] Filename to determine the format from.
- **content_type** (*str*) – [optional] Content type to determine the format from.

Returns
The converted spec and content type.

Return type
tuple

Raises

- *ParseError* – when parsing fails.

- *ConversionError* – when conversion fails.

prance.convert.**convert_url**(*url*, *cache={}*)

Fetch a URL, and try to convert it to OpenAPI 3.x.y.

Parameters
url (*str*) – The URL to fetch.

Returns
The converted spec and content type.

Return type
tuple

Raises

- *ParseError* – when parsing fails.
- *ConversionError* – when conversion fails.

4.4 prance.util.formats

This submodule contains file format related utility code for Prance.

4.4.1 Exceptions

exception prance.util.formats.**ParseError**

Bases: ValueError

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

4.4.2 Functions

prance.util.formats.**format_info**(*format_name*)

Return content type and extension for a supported format.

Valid formats are *YAML* or *JSON*.

Parameters
format_name (*str*) – The name of the format.

Returns
The preferred content type and file name extension, or (None, None) if the format name is not supported.

Return type
tuple

prance.util.formats.**parse_spec**(*spec_str*, *filename=None*, ***kwargs*)

Return a parsed dict of the given spec string.

The function exists for legacy reasons and just wraps parse_spec_details, returning only the parsed specs.

Parameters

- **spec_str** (*str*) – The specifications as string.

- **filename** (*str*) – [optional] Filename to determine the format from.
- **content_type** (*str*) – [optional] Content type to determine the format from.

Returns
The specifications.

Return type
dict

Raises
ParseError – when parsing fails.

prance.util.formats.**parse_spec_details**(*spec_str*, *filename=None*, ***kwargs*)

Return a parsed dict of the given spec string.

Also returned are the detected mime type and file name extension.

The default format is assumed to be JSON, but if you provide a filename, its extension is used to determine whether YAML or JSON should be parsed.

Parameters

- **spec_str** (*str*) – The specifications as string.
- **filename** (*str*) – [optional] Filename to determine the format from.
- **content_type** (*str*) – [optional] Content type to determine the format from.

Returns
The specifications, mime type, and extension.

Return type
tuple

Raises
ParseError – when parsing fails.

prance.util.formats.**serialize_spec**(*specs*, *filename=None*, ***kwargs*)

Return a serialized version of the given spec.

The default format is assumed to be JSON, but if you provide a filename, its extension is used to determine whether YAML or JSON should be parsed.

Parameters

- **specs** (*dict*) – The specifications as dict.
- **filename** (*str*) – [optional] Filename to determine the format from.
- **content_type** (*str*) – [optional] Content type to determine the format from.

Returns
The serialized specifications.

Return type
str

4.5 prance.util.fs

This submodule contains file system utilities for Prance.

4.5.1 Functions

prance.util.fs.**abspath**(*filename*, *relative_to=None*)

Return the absolute path of a file relative to a reference file.

If no reference file is given, this function works identical to *canonical_filename*.

Parameters

- **filename** (*str*) – The filename to make absolute.
- **relative_to** (*str*) – [optional] the reference file name.

Returns
The absolute path

Return type
str

prance.util.fs.**canonical_filename**(*filename*)

Return the canonical version of a file name.

The canonical version is defined as the absolute path, and all file system links dereferenced.

Parameters
filename (*str*) – The filename to make canonical.

Returns
The canonical filename.

Return type
str

prance.util.fs.**detect_encoding**(*filename*, *default_to_utf8=True*, ***kwargs*)

Detect the named file's character encoding.

If the first parts of the file appear to be ASCII, this function returns 'UTF-8', as that's a safe superset of ASCII. This can be switched off by changing the *default_to_utf8* parameter.

Parameters

- **filename** (*str*) – The name of the file to detect the encoding of.
- **default_to_utf8** (*bool*) – Defaults to True. Set to False to disable treating ASCII files as UTF-8.
- **read_all** (*bool*) – Keyword argument; if True, reads the entire file for encoding detection.

Returns
The file encoding.

Return type
str

prance.util.fs.**from_posix**(*fname*)

Convert a path from posix-like, to the platform format.

Parameters
fname (*str*) – The filename in posix-like format.

Returns
The filename in the format of the platform.

Return type
str

prance.util.fs.**is_pathname_valid**(*pathname*)

Test whether a path name is valid.

Returns
True if the passed pathname is valid on the current OS, False otherwise.

Return type
bool

prance.util.fs.**read_file**(*filename*, *encoding=None*)

Read and decode a file, taking BOMs into account.

Parameters

- **filename** (*str*) – The name of the file to read.
- **encoding** (*str*) – The encoding to use. If not given, detect_encoding is used to determine the encoding.

Returns
The file contents.

Return type
unicode string

prance.util.fs.**to_posix**(*fname*)

Convert a path to posix-like format.

Parameters
fname (*str*) – The filename to convert to posix format.

Returns
The filename in posix-like format.

Return type
str

prance.util.fs.**write_file**(*filename*, *contents*, *encoding=None*)

Write a file with the given encoding.

The default encoding is 'utf-8'. It's recommended not to change that for JSON or YAML output.

Parameters

- **filename** (*str*) – The name of the file to read.
- **contents** (*str*) – The file contents to write.
- **encoding** (*str*) – The encoding to use. If not given, detect_encoding is used to determine the encoding.

4.6 prance.util.iterators

This submodule contains specialty iterators over specs.

4.6.1 Functions

prance.util.iterators.**item_iterator**(*value*, *path=()*)

Return item iterator over the a nested dict- or list-like object.

Returns each item value as the second item to unpack, and a tuple path to the item as the first value - in that, it behaves much like viewitems(). For list like values, the path is made up of numeric indices.

Given a spec such as this:

```
spec = {
  'foo': 42,
  'bar': {
    'some': 'dict',
  },
  'baz': [
    { 1: 2 },
    { 3: 4 },
  ]
}
```

Here, (parts of) the yielded values would be:

item	path
[...]	('baz',)
{ 1: 2 }	('baz', 0)
2	('baz', 0, 1)

Parameters
value (*dict/list*) – The specifications to iterate over.

Returns
An iterator over all items in the value.

Return type
iterator

prance.util.iterators.**reference_iterator**(*specs*, *path=()*)

Iterate through the given specs, returning only references.

The iterator returns three values:

- The key, mimicking the behaviour of other iterators, although it will always equal '$ref'
- The value
- The path to the item. This is a tuple of all the item's ancestors, in sequence, so that you can reasonably easily find the containing item. It does not include the final '$ref' key.

Parameters
specs (*dict*) – The specifications to iterate over.

Returns
An iterator over all references in the specs.

Return type
iterator

4.7 prance.util.resolver

This submodule contains a JSON inlining reference resolver.

4.7.1 Classes

class prance.util.resolver.**RefResolver**(*specs*, *url=None*, ***options*)

Bases: object

Resolve JSON pointers/references in a spec by inlining.

resolve_references()

Resolve JSON pointers/references in the spec.

4.7.2 Functions

prance.util.resolver.**default_reclimit_handler**(*limit*, *parsed_url*, *recursions=()*)

Raise prance.util.url.ResolutionError.

4.8 prance.util.url

This submodule contains code for fetching/parsing URLs.

4.8.1 Exceptions

exception prance.util.url.**ResolutionError**

Bases: LookupError

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

4.8.2 Functions

prance.util.url.**absurl**(*url*, *relative_to=None*)

Turn relative file URLs into absolute file URLs.

This is necessary, because while JSON pointers do not allow relative file URLs, Swagger/OpenAPI explicitly does. We need to make relative paths absolute before passing them off to jsonschema for verification.

Non-file URLs are left untouched. URLs without scheme are assumed to be file URLs.

Parameters

- **url** (*str/tuple*) – The input URL.
- **relative_to** (*str/tuple*) – [optional] The URL to which the input URL is relative.

Returns
The output URL, parsed into components.

Return type
tuple

prance.util.url.**fetch_url**(*url*, *cache={}*, *encoding=None*, *strict=True*)

Fetch the URL and parse the contents.

Same as fetch_url_text(), but also parses the content and only returns the parse results.

Parameters

- **url** (*tuple*) – The url, parsed as returned by *absurl* above.
- **cache** (*Mapping*) – An optional cache. If the URL can be found in the cache, return the cache contents.
- **encoding** (*str*) – Provide an encoding for local URLs to override encoding detection, if desired. Defaults to None.

Returns
The parsed file.

Return type
dict

prance.util.url.**fetch_url_text**(*url*, *cache={}*, *encoding=None*)

Fetch the URL.

If the URL is a file URL, the format used for parsing depends on the file extension. Otherwise, YAML is assumed.

The URL may also use the *python* scheme. In this scheme, the netloc part refers to an importable python package, and the path part to a path relative to the package path, e.g. *python://some_package/path/to/file.yaml*.

Parameters

- **url** (*tuple*) – The url, parsed as returned by *absurl* above.
- **cache** (*Mapping*) – An optional cache. If the URL can be found in the cache, return the cache contents.
- **encoding** (*str*) – Provide an encoding for local URLs to override encoding detection, if desired. Defaults to None.

Returns
The resource text of the URL, and the content type.

Return type
tuple

prance.util.url.**split_url_reference**(*base_url*, *reference*)

Return a normalized, parsed URL and object path.

The reference string is a JSON reference, i.e. a URL with a fragment that contains an object path into the referenced resource.

The base URL is used as a reference point for relative references.

Parameters

- **base_url** (*mixed*) – A parsed URL.

- **reference** (*str*) – A JSON reference string.

Returns
The parsed absolute URL of the reference and the object path.

prance.util.url.**urlresource**(*url*)

Return the resource part of a parsed URL.

The resource part is defined as the part without query, parameters or fragment. Just the scheme, netloc and path remains.

Parameters
url (*tuple*) – A parsed URL

Returns
The resource part of the URL

Return type
str

4.9 prance.util.exceptions

This submodule contains helpers for exception handling.

4.9.1 Functions

prance.util.exceptions.**raise_from**(*klass*, *from_value*, *extra_message=None*)

4.10 prance.util.path

This module contains code for accessing values in nested data structures.

4.10.1 Functions

prance.util.path.**path_get**(*obj*, *path*, *defaultvalue=None*, *path_of_obj=()*)

Retrieve the value from obj indicated by path.

Like dict.get(), except:

- Any Mapping or Sequence is supported.
- Path is itself a Sequence; the first part is applied to the passed object, the second part to the value returned from this operation, and so forth recursively.

Parameters

- **obj** (*mixed*) – The Sequence or Mapping from which to retrieve values.
- **path** (*Sequence*) – A Sequence of zero or more key/index elements.
- **defaultvalue** (*mixed*) – If the value at the path does not exist and this parameter is not None, it is returned. Otherwise an error is raised.

prance.util.path.**path_set**(*obj*, *path*, *value*, ***options*)

Set the value in obj indicated by path.

Setter anologous to path_get() above.

As setting values is a write operation, this function optionally creates intermediate objects to ensure all elements of path can be dereferenced.

Parameters

- **obj** (*mixed*) – The Sequence or Mapping from which to retrieve values.
- **path** (*Sequence*) – A Sequence of zero or more key/index elements.
- **value** (*mixed*) – The value to set.
- **create** (*bool*) – [optional] Flag indicating whether to create intermediate values or not. Defaults to False.

CHAPTER

FIVE

CHANGES

CHAPTER

SIX

PRANCE 0.22.11.04.0

6.1 Features

- consolidate and unify openapi-spec-validator api usage (#132)
- drop dead pythons and upgrade builds for python 3.7 - 3.10 (#137)
- migrate from distutils.version to packaging.version (#138)

CHAPTER

SEVEN

PRANCE 0.21.8.0 (2021-08-06)

7.1 Features

- Initial translating parser to inline other specs to new names. (#101)
- replace pyyaml with ruamel.yaml for modern yaml support (#110)
- Adopt black as code formatter. (#113)

7.2 Bugfixes

- RefResolver will again accept and if instructed resolve references using the “python” URL scheme. (#104)

CHAPTER

EIGHT

PRANCE 0.21.2 (2021-05-18)

8.1 Bugfixes

- widen chardet pin to ease dependency hell for when others haven't updated to >4 (#98)

CHAPTER

NINE

V0.21.1 (2021-05-18)

- quickfix for a missed rst issue in readme

CHAPTER

TEN

V0.21.0 (2021-05-18)

10.1 Features

- Implement initial part of maintainer switch (#93)
  - @RonnyPfannschmidt is the new maintainer, plans to move to jazzband
  - License is now MIT after coordination with Jens
  - begin to use pre-commit + pyupgrade
  - set up for setuptools_scm as bumpversion breaks with normalized configfiles
  - github actions
  - modernize setup.py/cfg
- return to towncrier default templates

CHAPTER

ELEVEN

V0.20.2

- #83: Properly propagate strict mode down to nested resolvers.

CHAPTER

TWELVE

V0.20.1

Bugfix release:

- #85: Update dependencies, in particular chardet
- Miscellaneous: #86

CHAPTER

THIRTEEN

V0.20.0

- #77: Translate local references in external files by injecting them into the main specification.
- #78: Fix issue in RESOLVE_INTERNAL handling

CHAPTER

FOURTEEN

V0.19.0

- #72: Fix behaviour when attempting to resolve nonexistent local references: raise ResolutionError instead of what the OS provides.
- #69: Improve documentation with regards to JSON Schema and OpenAPI interoperability; some things are just not very well defined, and we make some strict assumptions in prance.
- Miscellaneous: #71

CHAPTER

FIFTEEN

V0.18.3

Bugfix release:

- #67: fix syntax warning.
- #69: when resolving references, if URL parsing fails, provide context on which URL was being parsed in error message.

CHAPTER

SIXTEEN

V0.18.2

Bugfix release:

- #65: fix error in resolving files only with ResolvingParser.

CHAPTER

SEVENTEEN

V0.18.1

Maintenance release, focusing on change requests from users.

- #23: Add support for partial resolution, i.e. resolving only internal references, local files, HTTP URLs, or any combination thereof.
- #36: Improve error handling by mentioning strict mode when openapi-spec-validator raises TypeError with very little context.
- #46: Reduce reliance on network in tests. Tests that require a network connection can now be skipped via "-m 'not requires_network'". Other tests have mocked connections.
- #55: RefResolver could set recursion limits, but the ResolvingParser did not pass related options on to the resolver. Fixed that. Also create & use reference cache in ResolvingParser.
- #60: Improve output when resolving references, by indicating the type of problem (missing key, index out of bounds) in the object or sequence where the error occurred.

CHAPTER

EIGHTEEN

V0.17.0

- #51: Try a lot more bytes when detecting file encoding. The new value is meant to be a multiple of sector/cluster size that's still reasonable on most OSes and volumes.
- #49: Remove Python 2.7 from supported/built versions. The CI vendors also don't love 3.4 any longer. Instead, we've added 3.7 and 3.8 where available.
- Miscellaneous: #53

CHAPTER

NINETEEN

V0.16.2

- #47: Fix deprecation warning by always preferring collections.abc over collections.

CHAPTER

TWENTY

V0.16.1

- #44: Add changelog generation via towncrier

PYTHON MODULE INDEX

p

prance, 11
prance.convert, 14
prance.mixins, 13
prance.util.exceptions, 23
prance.util.formats, 16
prance.util.fs, 18
prance.util.iterators, 20
prance.util.path, 23
prance.util.resolver, 21
prance.util.url, 21

INDEX

A

abspath() (*in module prance.util.fs*), 18
absurl() (*in module prance.util.url*), 21

B

BACKENDS (*prance.BaseParser attribute*), 12
BACKENDS (*prance.ResolvingParser attribute*), 13
BaseParser (*class in prance*), 12

C

CacheSpecsMixin (*class in prance.mixins*), 13
canonical_filename() (*in module prance.util.fs*), 18
ConversionError, 15
convert_spec() (*in module prance.convert*), 15
convert_str() (*in module prance.convert*), 15
convert_url() (*in module prance.convert*), 16

D

default_reclimit_handler() (*in module prance.util.resolver*), 21
detect_encoding() (*in module prance.util.fs*), 18

F

fetch_url() (*in module prance.util.url*), 22
fetch_url_text() (*in module prance.util.url*), 22
format_info() (*in module prance.util.formats*), 16
from_posix() (*in module prance.util.fs*), 18

I

is_pathname_valid() (*in module prance.util.fs*), 19
item_iterator() (*in module prance.util.iterators*), 20

J

json() (*prance.BaseParser method*), 12
json() (*prance.mixins.JSONMixin method*), 14
json() (*prance.ResolvingParser method*), 12
JSONMixin (*class in prance.mixins*), 13

M

module
    prance, 11
    prance.convert, 14
    prance.mixins, 13
    prance.util.exceptions, 23
    prance.util.formats, 16
    prance.util.fs, 18
    prance.util.iterators, 20
    prance.util.path, 23
    prance.util.resolver, 21
    prance.util.url, 21

P

parse() (*prance.BaseParser method*), 12
parse() (*prance.ResolvingParser method*), 13
parse_spec() (*in module prance.util.formats*), 16
parse_spec_details() (*in module prance.util.formats*), 17
ParseError, 16
path_get() (*in module prance.util.path*), 23
path_set() (*in module prance.util.path*), 23
prance
    module, 11
prance.convert
    module, 14
prance.mixins
    module, 13
prance.util.exceptions
    module, 23
prance.util.formats
    module, 16
prance.util.fs
    module, 18
prance.util.iterators
    module, 20
prance.util.path
    module, 23
prance.util.resolver
    module, 21
prance.util.url
    module, 21

R

raise_from() (*in module prance.util.exceptions*), 23

read_file() (*in module prance.util.fs*), 19
reference_iterator() (*in module prance.util.iterators*), 20
RefResolver (*class in prance.util.resolver*), 21
ResolutionError, 21
resolve_references() (*prance.util.resolver.RefResolver method*), 21
ResolvingParser (*class in prance*), 12

S

serialize_spec() (*in module prance.util.formats*), 17
SPEC_VERSION_2_PREFIX (*prance.BaseParser attribute*), 12
SPEC_VERSION_2_PREFIX (*prance.ResolvingParser attribute*), 13
SPEC_VERSION_3_PREFIX (*prance.BaseParser attribute*), 12
SPEC_VERSION_3_PREFIX (*prance.ResolvingParser attribute*), 13
specs_updated() (*prance.BaseParser method*), 12
specs_updated() (*prance.mixins.CacheSpecsMixin method*), 13
specs_updated() (*prance.mixins.JSONMixin method*), 14
specs_updated() (*prance.mixins.YAMLMixin method*), 14
specs_updated() (*prance.ResolvingParser method*), 13
split_url_reference() (*in module prance.util.url*), 22

T

to_posix() (*in module prance.util.fs*), 19

U

urlresource() (*in module prance.util.url*), 23

V

ValidationError, 11

W

with_traceback() (*prance.convert.ConversionError method*), 15
with_traceback() (*prance.util.formats.ParseError method*), 16
with_traceback() (*prance.util.url.ResolutionError method*), 21
with_traceback() (*prance.ValidationError method*), 11
write_file() (*in module prance.util.fs*), 19

Y

yaml() (*prance.BaseParser method*), 12
yaml() (*prance.mixins.YAMLMixin method*), 14
yaml() (*prance.ResolvingParser method*), 13
YAMLMixin (*class in prance.mixins*), 14